

# COTOBA Agent OSS による 対話システムの実装

2020/10/08 第1回対話システム勉強会  
(株) コトバデザイン  
プリンシパル・ソリューションアーキテクト  
山上 勝義

本発表の一部は、令和2年度総務省委託研究「高度対話エージェント技術の研究開発・実証」の成果によるものである。

# 講演者自己紹介

自然言語処理・音声インターフェース・対話システムの研究開発をやってきました

- 京都大学大学院工学部卒（修士）（1993/03）
- パナソニック（株）（1993/04 – 2019/01）
  - 日本語形態素解析技術の研究開発
  - テキスト音声合成技術の研究開発、製品化
  - 対話システムの研究開発、製品化（2014 ～ 2019/01）  
[https://www.jstage.jst.go.jp/article/pjsai/JSAI2017/0/JSAI2017\\_3Q16in1/\\_article/-char/ja](https://www.jstage.jst.go.jp/article/pjsai/JSAI2017/0/JSAI2017_3Q16in1/_article/-char/ja)
- 産総研 人工知能研究センター（出向）（2017/3 ～ 2018/12）
  - 知識ベースに基づくユーザ意図理解の対話技術の研究  
[https://www.isca-speech.org/archive/AI-MHRI\\_2018/pdfs/2.pdf](https://www.isca-speech.org/archive/AI-MHRI_2018/pdfs/2.pdf)
- （株）コトバデザイン（2019/2 ～ ）
  - 対話エージェント応用のソリューション開発

# 本日正式紹介する内容

- COTOBA Agent OSS について
- COTOBA Agent OSS の動かし方
- AIML での対話シナリオの実装例
  - 一問一答型
  - ネットワーク型
  - スロットフィリング型
  - 他の対話エンジンとの連携

# COTOBA Agent OSS について

# COTOPA Agent OSS: GitHub



<http://bit.ly/cotobaoss>

Search or jump to... Pull requests Issues Marketplace Explore

cotobadesign / cotoba-agent-oss

Unwatch 11 **★ Unstar 65** Fork 7

<> Code Issues 9 Pull requests 1 Actions Projects Wiki Security Insights Settings

master 6 branches 6 tags

Go to file Add file Code

cliffCotobadesign Merge pull request #26 from cotobadesign/feature/json\_element... 78e5c2c on 4 Aug 68 commits

File/Folder	Description	Time
.github/ISSUE_TEMPLATE	Add issue template.	3 months ago
dialogue-engine	Added a pattern to the json format check when deploying AIML for JS...	2 months ago
nlu	Fix README	6 months ago
.gitignore	add android,ios and python sample client for COTOPA Agent.	3 months ago
LICENSE	improvement debug information	3 months ago
README.md	modified url	6 months ago
README_ja.md	modified url	6 months ago

README.md

## COTOPA Agent dialog engine

About

No description, website, or topics provided.

Readme

MIT License

Releases

6 tags

Create a new release

Packages

No packages published

Publish your first package

Contributors 4

Star 登録  
よろしく  
お願いします！

# COTOBA Agent OSS の諸元

- インタプリタ型の対話エンジン（対話処理実行環境）です  
対話エンジン自体は Python で記述されています
- AIML (Artificial Intelligence Markup Language)で対話処理を記述します  
[https://ja.wikipedia.org/wiki/Artificial\\_Intelligence\\_Markup\\_Language](https://ja.wikipedia.org/wiki/Artificial_Intelligence_Markup_Language)  
COTOBA DESIGN が AIML の仕様を一部独自拡張しています  
- JSON データ構造の取り扱い／REST API での外部サービス呼び出しなど  
⇒ COTOBA AIML (cAIML) と呼んでいます  
ドキュメント: <https://cotoba-agent-oss-docs-ja.readthedocs.io/en/latest/index.html>
- 日本語／英語を入出力にした対話処理を実行出来ます  
Mecab による形態素解析を利用しています
- 非言語情報（メタデータ）入出力もサポートしています  
センサデータやロボットと連携した対話処理も記述できます  
2020年音響学会春季研究発表会: [https://jglobal.jst.go.jp/detail?JGLOBAL\\_ID=202002229716304969](https://jglobal.jst.go.jp/detail?JGLOBAL_ID=202002229716304969)

# COTOBA Agent OSS でできること(1) COTOBA DESIGN

## □ ルールベースの対話処理を実行できます

### □ ルールベースの対話処理とは？

- if 入力テキストがある条件を満たす then ある応答テキストを出力する
- パターンマッチによる条件分岐処理を組み合わせることで処理内容を記述
- Python 風にと書くと

こんにちは

ごきげんよう

#### COTOBA Agent OSS

```
def rule_base(in_text):  
    if in_text == "こんにちは":  
        out_text = "ごきげんよう"  
    elif in_text == "ありがとう":  
        out_text = "どういたしまして"  
    else:  
        out_text = "わかりませんでした"  
    return out_text
```

ありがとう

どういたしまして

# COTOBA Agent OSS の動かし方



# 事前準備

## □ MeCab (日本語形態素解析ツール) のインストール

macOS の場合: homebrew でインストール

```
$ brew install mecab  
$ brew install mecab-ipadic
```

Linux (Ubuntu) の場合: apt でインストール

```
$ sudo apt install mecab  
$ sudo apt install libmecab-dev  
$ sudo apt install mecab-ipadic-utf8
```

Windows の場合:

<https://github.com/ikegami-yukino/mecab/releases/tag/v0.996>

を参考にインストールして下さい

# COTOPA Agent OSS のセットアップ



## □ COTOPA Agent OSS のインストール

※ 動作確認済みの Python のバージョンは 3.7.X です

```
$ git clone https://github.com/cotobadesign/cotoba-agent-oss.git
Cloning into 'cotoba-agent-oss'...
...
$ cd cotoba-agent-oss/dialog-engine
$ pip3 install -r requirements.txt
...
$
```

# Python の仮想環境でセットアップ

## □ macOS の場合:

python3, pyenv, pipenv のインストール

```
$ brew install python3  
$ brew install pyenv  
$ pip3 install pipenv
```

Python 仮想環境の構築 (git clone <https://github.com/cotobadesign/cotoba-agent-oss.git> の後)

```
$ cd cotoba-agent-oss/dialog-engine  
$ pipenv --python 3.7 install  
$ pipenv shell  
(dialog-engine) $
```

## □ Linux (Ubuntu) の場合:

参考 : <https://note.com/mokuichi/n/n627eb6773aad>

# COTOBA Agent OSS の起動（１）

## □ コンソールからの起動 (basic/storage/categories/ にあるサンプルの AIMLを実行)

```
$ cd cotoba-agent-oss/dialog-engine/basic/script
$ ./console.sh
Loading, please wait...
/Users/yamagami/cotoba-agent-oss/dialog-engine/basic/script
basic, vv0.0.1, initiated March 01, 2020
That's a difficult question.
>>> good morning
Are you sleeping without staying up late?
>>> yes
It's nice to wake up in the morning.
>>> おはよう
昨日は、早く寝ましたか？
>>> はい
朝は、すっきり目が覚めることができたんですね。
>>> What time is it now
14:19:08
>>> 今何時
14時19分13秒です。
>>>
```

# COTOBA Agent OSS の起動 (2)

## □ REST API サーバとしての起動 (basic/storage/categories/ にあるサンプルの AIMLを実行)

```
$ cd cotoba-agent-oss/dialog-engine/basic/script
$ ./flaskYadlan.sh
Initiating Yadlan Flask Service...
Loading, please wait...
Server start...
* ...
```

## □ サンプルクライアントからの接続

```
$ cd cotoba-agent-oss/dialog-engine/basic/script
$ python3 simple_request.py
>>> おはよう
夜更かしせずに寝ていますか？
>>> はい
朝は、すっきり目が覚めることができたんですね。
>>>
```

# 対話シナリオのファイル群

```
$ cd cotoba-agent-oss/dialog-engine/basic
$ tree storage
storage
├── categories
│   ├── basic_en.aiml
│   └── basic_ja.aiml
├── lookups
│   ├── denormal.txt
│   ├── gender.txt
│   ├── normal.txt
│   ├── person.txt
│   └── person2.txt
├── maps
│   ├── downcount.txt
│   └── upcount.txt
└── ...
```

シナリオファイル

置換処理定義ファイル

map タグ用定義ファイル

```
storage
├── ...
└── ...
    ├── properties
    │   ├── nlu_servers.yaml
    │   ├── defaults.txt
    │   └── properties.txt
    └── sets
        ├── daytimeAmbiguous.txt
        ├── daytimeAmbiguous_en.txt
        └── daytimeBad.txt
        ...
```

NLUサーバ設定値定義

name 変数初期値定義

bot タグ属性値定義

set タグ用定義ファイル

自作した AIML を試す場合

- \$ cp -r basic sample などとしてsample/storage/categories/ に \*.aiml をおく (使わない \*.aiml は削除)
- \$ cd sample/script && ./console.sh で実行

# cAIML での対話シナリオの実装例

# AIMLの基本構造

```
<?xml version="1.0" encoding="UTF-8" ?>
<aiml version="2.0">
```

```
<category>
```

```
<pattern>こんにちは</pattern>
```

```
<template>ごきげんよう</template>
```

```
</category>
```

```
<category>
```

```
<pattern>ありがとう</pattern>
```

```
<template>どういたしまして</template>
```

```
</category>
```

```
<category>
```

```
<pattern>*</pattern>
```

```
<template>わかりませんでした</template>
```

```
</category>
```

```
</aiml>
```

## □ 3つのタグで1つのルールを表現します

**category** タグ: 1つのルール

**pattern** タグ: 当該ルールを適用する入力文のパターン

**template** タグ: 当該ルールを適用した時の出力文

[Docs: 4. COTOBA AIML: 基本要素](#)

## □ 入力文が pattern タグ内に指定したパターンと照合され、適用されるルールが決定されます

**pattern** タグ内に記述したパターンの種類によって

マッチングの優先順位が規定されています

※記述した順番でマッチングが行われるわけではありません

[Docs: 7. パターンマッチング](#)

## □ パターンとして通常の文字列以外に任意の文字列と照合するワイルドカードが指定できます (例: \*)

[Docs: 7.1. \\* ワイルドカード](#)

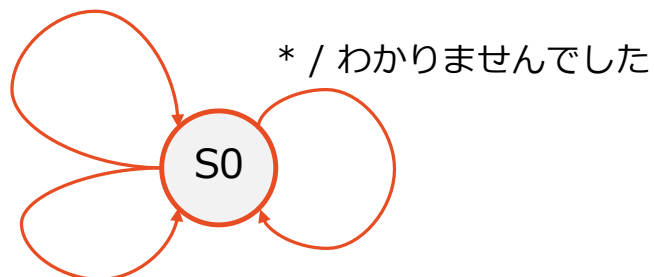


# ルールベースの対話制御モデルの種類

## □ 一問一答型

内部状態の状態遷移無し（内部状態は1つ）

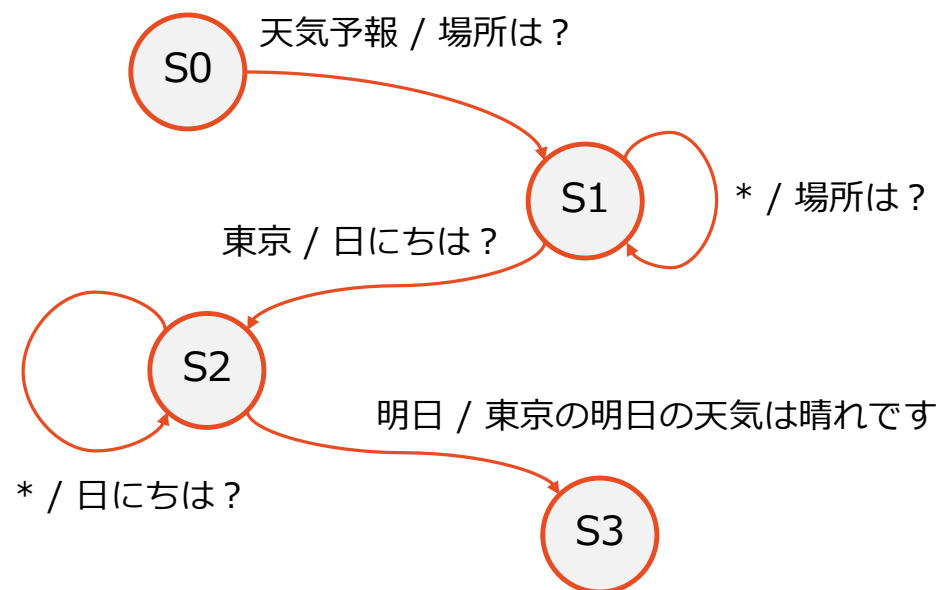
こんにちは / ごきげんよう



ありがとう / どういたしまして

## □ ネットワーク型

内部状態の状態遷移有り（内部状態は複数）



凡例：入力文 / 出力文

# 一問一答型の実装例

## sample1.aiml

```
<?xml version="1.0" encoding="UTF-8" ?>
<aiml version="2.0">

  <category>
    <pattern>こんにちは</pattern>
    <template>ごきげんよう</template>
  </category>

  <category>
    <pattern>ありがとう</pattern>
    <template>どういたしまして</template>
  </category>

  <category>
    <pattern>*</pattern>
    <template>「 <star /> 」はわかりませんでした</template>
  </category>

</aiml>
```

- ワイルドカードにマッチした文字列を template タグ中で参照(取得)出来ます

star タグ: <star />

[Docs: 6.1.57. star](#)

- 対話例

```
$ cd cotoba-agent-oss/dialog-engine/basic/script
$ ./console.sh
>>> こんにちは
ごきげんよう。
>>> ありがとう
どういたしまして。
>>> さようなら
「さようなら」はわかりませんでした。
>>>
```

# ネットワーク型の実装例 (1/3)

sample2.aiml

```
<?xml version="1.0" encoding="UTF-8" ?>
<aiml version="2.0">
```

```
  <category>
    <pattern>天気予報</pattern>
    <template>
      <think>
        <set name="topic">S1</set>
      </think>
      場所は？
    </template>
  </category>
```

↓ 次ページに続きます

- ネットワーク型の対話制御では複数の内部状態の **状態遷移** を表現する必要がある。。。

**topic** タグを使って **category** タグのルールが適用されるコンテキスト(条件)を指定出来ます

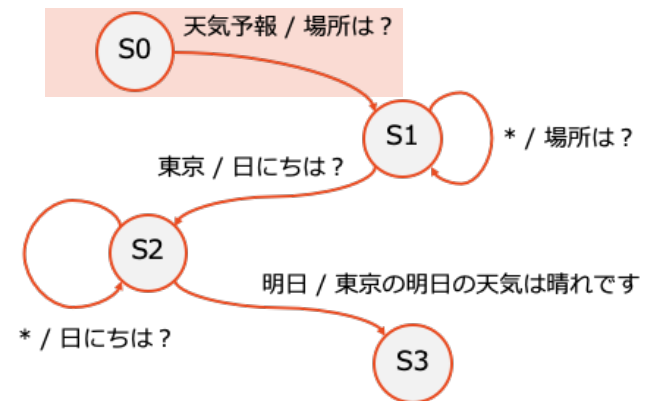
[Docs: 4.5 topic](#)

ここで **topic** に 'S1' (内部状態) を設定しています  
⇒ '天気予報' が入力されたら 'S1' に状態遷移

- **think** タグで囲まれた記述は出力文に反映されません

[Docs: 6.1.61. think](#)

**think** タグで囲わないとどうなる？  
⇒ 出力文が 'S1場所は？' になってしまいます



# ネットワーク型の実装例 (2/3)

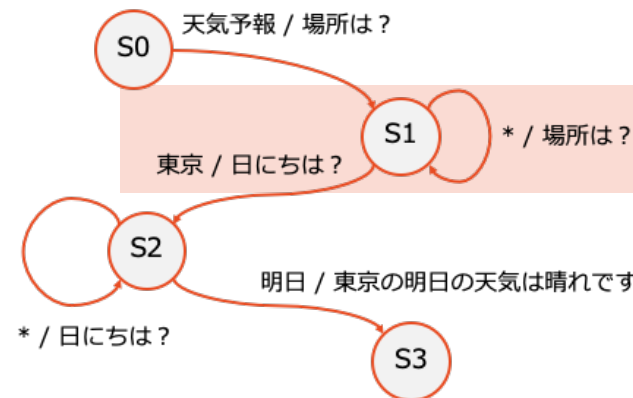
↓ 前ページからの続きです

```
<topic name="S1">
  <category>
    <pattern>東京</pattern>
    <template>
      <think>
        <set name="topic">S2</set>
      </think>
      日にちは？
    </template>
  </category>
  <category>
    <pattern>*</pattern>
    <template>場所は？</template>
  </category>
</topic>
```

□ topic タグで囲まれた category は、topic が name で指定された値('S1')の時のみ適用されます

□ '東京' が入力されたら 'S2' に状態遷移して '日にちは？' を出力

□ '東京' 以外が入力されたら状態遷移せず '場所は？' を出力



↓ 次ページに続きます

# ネットワーク型の実装例 (3/3)

↓ 前ページからの続きです

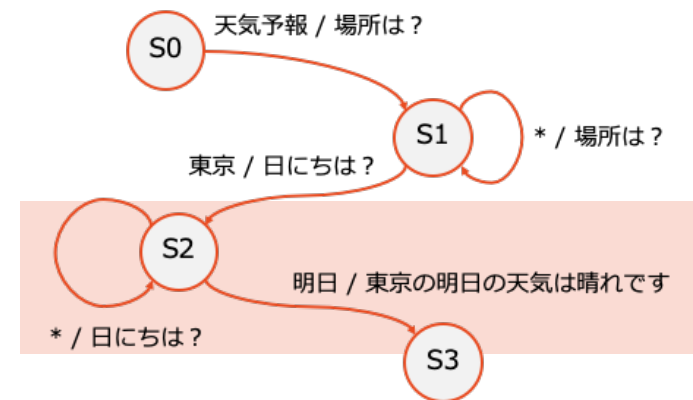
```
<topic name="S2">
  <category>
    <pattern>明日</pattern>
    <template>
      <think>
        <set name="topic">S3</set>
      </think>
      東京の明日の天気は晴れです
    </template>
  </category>
  <category>
    <pattern>*</pattern>
    <template>日にちは？</template>
  </category>
</topic>
```

</aiml>

□ **topic** タグで囲まれた **category** は、**topic** が **name** で指定された値('S2')の時のみ適用されます

□ '明日' が入力されたら 'S3' に状態遷移して '東京の明日の天気は晴れです' を出力

□ '明日' 以外が入力されたら状態遷移せず '日にちは？' を出力



□ **topic** の設定値をデフォルト値に戻す (<set name="topic">\*</set>) と **topic** タグで囲まれていない **category** が適用されます

# ネットワーク型の実装例 (3/4)

## □ 対話例

```
$ cd cotoba-agent-oss/dialog-engine/sample2/script
$ ./console.sh
>>> 天気予報
場所は？
>>> こんにちは
場所は？
>>> 東京
日にちは？
>>> 台風が心配です
日にちは？
>>> 明日
東京の明日の天気は晴れです
```

# AIML の変数

## sample3.aiml

```
<?xml version="1.0" encoding="UTF-8" ?>
<aiml version="2.0">
  <category>
    <pattern>私の名前は * です</pattern>
    <template>
      <think> <set name="user_name"><star /></set> </think>
      <get name="user_name" /> さん、こんにちは！
    </template>
  </category>
  <category>
    <pattern>私の出身地は * です</pattern>
    <template>
      <get name="user_name" />さんは
      <set var="birth_place" ><star /></set> の生まれなんですね。
      <get var="birth_place" /> は良いところですね。
    </template>
  </category>
</aiml>
```

### □ set タグ: 変数への値の代入

<set xxx="変数名"> 値 </set>

### □ get タグ: 変数の値の参照

<get xxx="変数名" />

### □ ローカル変数とグローバル変数があります

ローカル変数: **var="変数名"** で指定

グローバル変数: **name="変数名"** で指定

ローカル変数のスコープは  
**当該 category の中のみ**

グローバル変数のスコープは  
**すべての category の中**

[Docs: 1.4. 対話内容の抽出、変数利用](#)

# スロットフィリング型の対話 (1/8)

## □ スロットフィリング型の対話とは？

- ユーザから順次必要な情報を聞き出し、すべての情報が取得出来たらタスクを実行
- 必要な情報をスロットとして定義、ユーザの発話から取得した情報でスロットを埋める

## □ レストラン予約タスクの場合

日にち (今日, 明日, ...), 時間 (18時, 19時, ...), 人数 (2人, 3人, ...) の3つのスロットを定義

U: レストラン予約

S: 予約する日にち、時間、人数、を教えてください

U: 日にちは明日です

S: 予約する時間、人数、を教えてください

U: 人数は4人です

S: 予約する時間、を教えてください

U: 時間は19時です

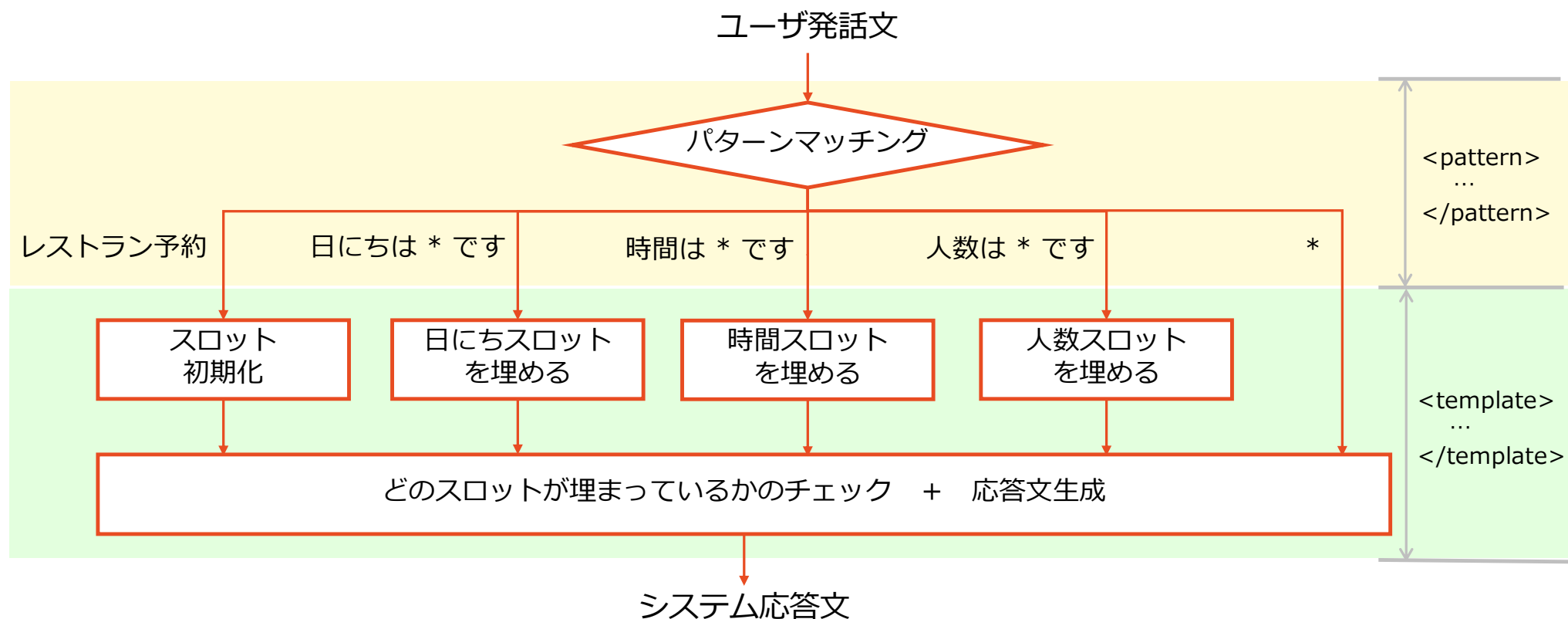
S: 明日、19時、4人で予約します

ユーザが  
日にち、時間、人数を発話する順番は  
この対話例の通りで無くてもOK！



# スロットフィリング型の対話 (2/8)

## □ スロットフィリング型の対話の処理フロー



# スロットフィリング型の対話 (3/8)

sample4.aiml

```
<?xml version="1.0" encoding="UTF-8" ?>
<aiml version="2.0">
  <category>
    <pattern>レストラン予約</pattern>
    <template>
      <think>
        <set name="day" >unknown</set>
        <set name="time" >unknown</set>
        <set name="num_of_person" >unknown</set>
      </think>
      <srai>__RESPONSE__</srai>
    </template>
  </category>
```

## □ スロットの表現

day: 日にちのスロットの変数

time: 時間のスロットの変数

num\_of\_person: 人数のスロットの変数

各スロットの変数を初期化 ('unknown')

## □ think タグに囲まれた部分は出力文に反映せず

[Docs: 6.1.61. think](#)

## □ srai タグ: <srai> 文字列 </srai>

'文字列' でパターンマッチを再実行する

[Docs: 6.1.55. srai](#)

↓ 次ページに続きます

スロットチェック+応答文生成の処理 (category) を呼び出し (後述)

```
<category>
  <pattern>__RESPONSE__</pattern>
  <template> ... </template>
</category>
```

# スロットフィリング型の対話 (4/8)

↓ 前ページからの続きです

```
<category>
  <pattern>日にちは * です</pattern>
  <template>
    <think>
      <set name="day"><star /></set>
    </think>
    <srai>__RESPONSE__</srai>
  </template>
</category>
```

□ star タグ: <star />

ワイルドカードにマッチした文字列を  
**template** タグ中で参照(取得)出来ます

[Docs: 6.1.57. star](#)

□ スロット変数 day を埋める処理

□ 応答文生成処理を呼び出し

```
<category>
  <pattern>時間は * です</pattern>
  <template>
    <think>
      <set name="time"><star /></set>
    </think>
    <srai>__RESPONSE__</srai>
  </template>
</category>
```

□ スロット変数 time を埋める処理

□ 応答文生成処理を呼び出し

↓ 次ページに続きます

# スロットフィリング型の対話 (5/8)

↓ 前ページからの続きです

```
<category>
  <pattern>人数は * です</pattern>
  <template>
    <think>
      <set name="num_of_person"><star /></set>
    </think>
    <srai>__RESPONSE__</srai>
  </template>
</category>
```

□ スロット変数 num\_of\_person を埋める処理

□ 応答文生成処理を呼び出し

```
<category>
  <pattern> * </pattern>
  <template>
    「 <star /> 」はわかりません。
    <srai>__RESPONSE__</srai>
  </template>
</category>
```

□ 想定外の発話入力の際の処理

↓ 次ページに続きます

# スロットフィリング型の対話 (6/8)

↓ 前ページからの続きです

```
<category>
  <pattern>__RESPONSE__</pattern>
  <template>
    <think>
      <set var="response">
        <condition name="day">
          <li value="unknown">日にち、</li>
        </condition>
        <condition name="time">
          <li value="unknown">時間、</li>
        </condition>
        <condition name="num_of_person">
          <li value="unknown">人数、</li>
        </condition>
      </set>
    </think>
```

## □ condition タグ: 条件分岐処理

変数の値によってどの文字列を返すかを制御

[Docs: 6.1.7.2. 条件分岐](#)

```
<condition xxx="変数名">
  <li value="値1"> 文字列1 </li>
  <li value="値2"> 文字列2 </li>
  ...
  <li> 文字列3 </li>
</condition>
```

if ~ elif ~ else 構文に相当

- 各スロット変数の値が埋まっているか ('unknown' 以外か)どうかで、変数 response に設定する文字列を制御  
⇒ すべてのスロット変数の値が埋まっていれば response は 'unknown'

↓ 次ページに続きます

# スロットフィリング型の対話 (7/8)

↓ 前ページからの続きです

```
<condition var="response">
  <li value="unknown">
    <get name="day" />、
    <get name="time" />、
    <get name="num_of_person" /> で予約します
  </li>
  <li>
    <get var="response" /> を教えてください
  </li>
</condition>
</template>
</category>

</aiml>
```

□ response が未定義('unknown')ならば  
スロット変数で応答文を生成  
⇒ レストラン予約の実行を通知

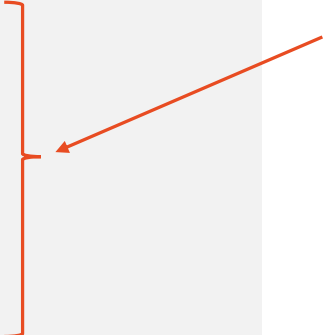
□ response が未定義でなければ  
response を使って応答文を生成  
⇒ 埋まっていないスロットの情報を  
ユーザに要求

# スロットフィリング型の対話 (8/8)

## □ 対話例

```
$ cd cotoba-agent-oss/dialog-engine/sample4/script
$ ./console.sh
>>> レストラン予約
日にち、時間、人数、を教えてください
>>> 人数は4人です
日にち、時間、を教えてください
>>> 時間は19時です
日にち、を教えてください
>>> 日にちは明日です
明日、19時、4人で予約します
>>>
```

日にち、時間、人数の発話の順序は任意



# 他の対話エンジンとの連携 (1/4)

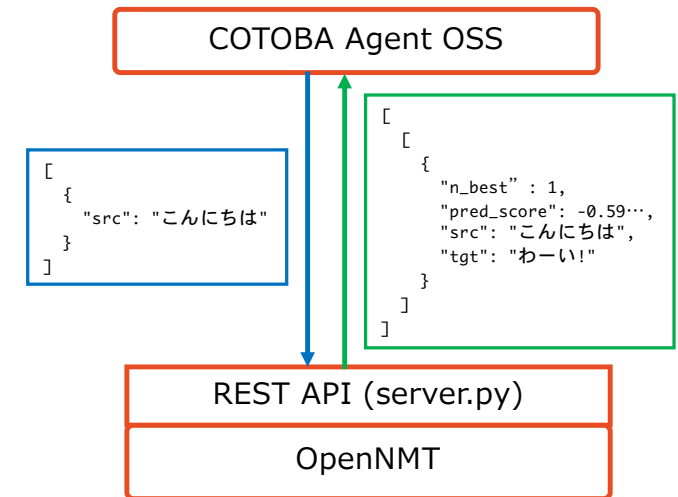
■ **sraix** タグ: AIML から REST API を呼び出す [Docs: 13. SubAgent](#)

```
<sraix>
  <host> REST API エンドポイント URL </host>
  <method>POST</method>
  <header> HTTP リクエストヘッダ情報 </header>
  <body> リクエストデータJSON </body>
</sraix>
```

レスポンスのJSONはローカル変数 `__SUBAGENT_BODY__` に格納されます

■ **json** タグ: 変数の JSON データの値の参照・値の代入 [Docs: 9. JSON](#)

	値の参照	値の代入
リスト	<pre>&lt;json var="var1" index="0" /&gt; var1 = ["a", "b", "c"]</pre>	<pre>&lt;json var="var1" index="1"&gt;B&lt;/json&gt; var1 = ["a", "B", "c"]</pre>
連想配列	<pre>&lt;json var="var2.key1" /&gt; var2 = {"key1": "a", "key2": "b"}</pre>	<pre>&lt;json var="var2.key2"&gt;B&lt;/json&gt; var2 = {"key1": "a", "key2": "B"}</pre>





## 他の対話エンジンとの連携 (2/4)

sample5.aiml

```
<?xml version="1.0" encoding="UTF-8" ?>
<aiml version="2.0">
  <category>
    <pattern>レストラン予約</pattern> ...
  </category>
  <category>
    <pattern>日にちは * です</pattern> ...
  </category>
  <category>
    <pattern>時間は * です</pattern> ...
  </category>
  <category>
    <pattern>人数は * です</pattern> ...
  </category>
  <category>
    <pattern>__RESPONSE__</pattern> ...
  </category>
```

sample4.aiml と同じコードです

↓ 次ページに続きます

# 他の対話エンジンとの連携 (3/4)

↓ 前ページからの続きです

```
<category>
  <pattern> * </pattern>
  <template>
    <think>
      <sraix>
        <host>http://localhost:5000/translator/translate</host>
        <method>POST</method>
        <header>"Content-type":"application/json;charset=UTF-8"</header>
        <body>[ {"src": "<star />", "id": 0} ]</body>
      </sraix>
      <set var="first_item"><json var="__SUBAGENT_BODY__" index="0" /></set>
      <set var="first_item_in_first_item">
        <json var="first_item" index="0" />
      </set>
      <set var="tgt"><json var="first_item_in_first_item.tgt" /></set>
    </think>
    <get var="tgt" />
  </template>
</category>

</aiml>
```

OpenNMT の  
REST API の呼び出し

OpenNMT のレスポンスから  
応答文を取り出し

# 他の対話エンジンとの連携 (4/4)

## □ 対話例

```
$ cd cotoba-agent-oss/dialog-engine/sample5/script
$ ./console.sh
>>> レストラン予約
日にち、時間、人数、を教えてください
>>> 日にちは今日です
時間、人数、を教えてください
>>> 人数は4人です
時間、を教えてください
>>> 時間は17時です
今日、17時、4人で予約します
>>> ありがとう
ええんやで __BR__ ありがと
>>>
```

AIML のスロットフィリング型対話

sraix で呼び出した OpenNMT の雑談対話

Thank you for your attention.